EMPOWERING GRAPH REPRESENTATION LEARNING WITH TEST-TIME GRAPH TRANSFORMATION

Wei Jin\textsuperscript{1}, Tong Zhao\textsuperscript{2}, Jiayuan Ding\textsuperscript{1}, Yozen Liu\textsuperscript{2}, Jiliang Tang\textsuperscript{1} and Neil Shah\textsuperscript{2}
\textsuperscript{1}Michigan State University \hspace{1em} \textsuperscript{2}Snap Inc.
{\{jinwei2,dingjia5,tangjili\}@msu.edu, \{tzhao,yliu2, nshah\}@snap.com}

ABSTRACT

As powerful tools for representation learning on graphs, graph neural networks (GNNs) have facilitated various applications from drug discovery to recommender systems. Nevertheless, the effectiveness of GNNs is immensely challenged by issues related to data quality, such as distribution shift, abnormal features and adversarial attacks. Recent efforts have been made on tackling these issues from a modeling perspective which requires additional cost of changing model architectures or re-training model parameters. In this work, we provide a data-centric view to tackle these issues and propose a graph transformation framework named GT\textsuperscript{RANS} which adapts and refines graph data at test time to achieve better performance. We provide theoretical analysis on the design of the framework and discuss why adapting graph data works better than adapting the model. Extensive experiments have demonstrated the effectiveness of GT\textsuperscript{RANS} on three distinct scenarios for eight benchmark datasets where suboptimal data is presented. Remarkably, GT\textsuperscript{RANS} performs the best in most cases with improvements up to 2.8\%, 8.2\% and 3.8\% over the best baselines on three experimental settings.

1 INTRODUCTION

Graph representation learning has been at the center of various real-world applications, such as drug discovery (Duvenaud et al., 2015; Guo et al., 2022), recommender systems (Ying et al., 2018; Pan et al., 2019; Sankar et al., 2021), forecasting (Yang et al., 2020; Derrow-Pinion et al., 2021) and outlier detection (Zhao et al., 2021b; Deng & Hooi, 2021). In recent years, there has been a surge of interest in developing graph neural networks (GNNs) as powerful tools for graph representation learning (Kapl & Welling, 2016a; Veličković et al., 2018; Hamilton et al., 2017; Wu et al., 2019). Remarkably, GNNs have achieved state-of-the-art performance on numerous graph-related tasks including node classification, graph classification and link prediction (Chien et al., 2021; You et al., 2021; Zhao et al., 2022b).

Despite the enormous success of GNNs, recent studies have revealed that their generalization and robustness are immensely challenged by the data quality (Jin et al., 2021b; Li et al., 2022). In particular, GNNs can behave unreliably in scenarios where sub-optimal data is presented:

1. Distribution shift (Wu et al., 2022a; Zhu et al., 2021a). GNNs tend to yield inferior performance when the distributions of training and test data are not aligned (due to corruption or inconsistent collection procedure of test data).
2. Abnormal features (Li et al., 2021a). GNNs suffer from high classification errors when data contains abnormal features, e.g., incorrect user profile information in social networks.
3. Adversarial structure attack (Zügner et al., 2018; Li et al., 2021). GNNs are vulnerable to imperceptible perturbations on the graph structure which can lead to severe performance degradation.

To tackle these problems, significant efforts have been made on developing new techniques from the modeling perspective, e.g., designing new architectures and employing adversarial training strategies (Xu et al., 2019; Wu et al., 2022a). However, employing these methods in practice may be infeasible, as they require additional cost of changing model architectures or re-training model parameters, especially for well-trained large-scale models. The problem is further exacerbated when

\*Work done while author was on internship at Snap Inc.
adopting these techniques for multiple architectures. By contrast, this paper seeks to investigate approaches that can be readily used with a wide variety of pre-trained models and test settings for improving model generalization and robustness. Essentially, we provide a data-centric perspective to address the aforementioned issues by modifying the graph data presented at test-time. Such modification aims to bridge the gap between training data and test data, and thus enable GNNs to achieve better generalization and robust performance on the new graph. Figure 1 visually describes this idea: we are originally given with a test graph with abnormal features where multiple GNN architectures yield poor performance; however, by transforming the graph prior to inference (at test-time), we enable these GNNs to achieve much higher accuracy.

In this work, we aim to develop a data-centric framework that transforms the test graph to enhance model generalization and robustness, without altering the pre-trained model. In essence, we are faced with two challenges: (1) how to model and optimize the transformed graph data, and (2) how to formulate an objective that can guide the transformation process. First, we model the graph transformation as injecting perturbation on the node features and graph structure, and optimize them alternatively via gradient descent. Second, inspired by the recent progress of contrastive learning, we propose a parameter-free surrogate loss which does not affect the pre-training process while effectively guiding the graph adaptation. Our contributions can be summarized as follows:

1. For the first time, we provide a data-centric perspective to improve the generalization and robustness of GNNs with test-time graph transformation.
2. We establish a novel framework GTRANS for test-time graph transformation by jointly learning the features and adjacency structure to minimize a proposed surrogate loss.
3. Our theoretical analysis provides insights on what surrogate losses we should use during test-time graph transformation and sheds light on the power of data-adaptation over model-adaptation.
4. Extensive experimental results on three settings (distribution shift, abnormal features and adversarial structure attacks) have demonstrated the superiority of test-time graph transformation. Particularly, GTRANS performs the best in most cases with improvements up to 2.8%, 8.2% and 3.8% over the best baselines on three experimental settings.

Moreover, we note: (1) GTRANS is flexible and versatile. It can be equipped with any pre-trained GNNs and the outcome (the refined graph data) can be deployed with any model given its favorable transferability. (2) GTRANS provides a degree of interpretability, as it can show which kinds of graph modifications can help improve performance by visualizing the data.

2 RELATED WORK

**Distribution shift in GNNs.** GNNs have revolutionized graph representation learning and achieved state-of-the-art results on diverse graph-related tasks [Kipf & Welling, 2016a; Velickovic et al., 2018; Hamilton et al., 2017; Chien et al., 2021; Klicpera et al., 2018; Wu et al., 2019]. However, recent studies have demonstrated that GNNs yield sub-optimal performance on out-of-distribution data for node classification (Zhu et al., 2021a; Wu et al., 2022a; Gu et al., 2022) and graph classification (Chen et al., 2022; Buffelli et al., 2022; Li et al., 2022). These studies have introduced solutions to tackle distribution shifts. For example, EERM (Wu et al., 2022a) attributes the cause of distribution shifts to an unknown environmental variable, and trains GNNs under multiple environments produced by a generator. For a thorough review, we refer the readers to a recent survey [Li et al., 2022]. Unlike existing works, we target modifying the inputs via test-time adaption.

**Robustness of GNNs.** Recent studies have demonstrated the vulnerability of GNNs to graph adversarial attacks (Zügner et al., 2018; Zügner & Günnemann, 2019; Xu et al., 2019; Geisler et al., 2021), i.e., small perturbations on the input graph can mislead GNNs into making wrong predic-

---

**Figure 1:** We study the test-time graph transformation problem, which seeks to learn a refined graph such that pre-trained GNNs can perform better on the new graph compared to the original. Shown: An illustration of our proposed approach’s empirical performance on transforming a noisy graph.
tions. Several works make efforts towards developing new GNNs or adversarial training strategies to defend against attacks \cite{Xu2019, Zhu2019, Jin2021, Jin2021a, Jin2021b, Zhang2021, Zhang2022}. Instead of altering model training behavior, our work aims to modify the test graph to correct adversarial patterns.

**Graph Structure Learning & Graph Data Augmentation.** Graph structure learning and graph data augmentation both aim to improve GNNs’ generalization performance by augmenting the (training) graph data \cite{Zhao2022a, Jin2020, Chen2020, Zhao2021b} or perturbing the graph in a rule-based way \cite{Rong2020, Feng2020, Ding2022}. While our work also modifies the graph data, we focus on modifying the test data and not impacting the model training process.

**Test-time Training.** Our work is also related to test-time training \cite{Sun2020, Wang2021, Liu2021b, Zhang2021, Zhang2022}, which has raised a surge of interest in computer vision recently. To handle out-of-distribution data, \cite{Sun2020} propose the pioneer work of test-time training (TTT) by optimizing feature extractor via an auxiliary task loss. However, TTT alters training to jointly optimize the supervised loss and auxiliary task loss. To remove the need for training an auxiliary task, Tent \cite{Wang2021} proposes to minimize the prediction entropy at test-time. It is worth noting that Tent works by adapting the statistics and parameters in batch normalization (BN) layers, which may not always be employed by modern GNNs. In this work, we focus on a novel perspective of adapting the test graph data instead of the model, which makes no assumptions about the particular training procedure or architecture.

## 3 Methodology

We start by introducing the general problem of test-time graph transformation (TTGT). While our discussion mainly focuses on the node classification task where the goal is to predict the labels of nodes in the graph, it can be easily extended to other tasks. Consider that we have a training graph \(G_T\) and a test graph \(G_T\), and the corresponding set of node labels are denoted as \(\mathcal{Y}_T\) and \(\mathcal{Y}_T\), respectively. Note that the node sets in \(G_T\) and \(G_T\) can either be disjoint or overlapping, and they are not necessarily drawn from the same distribution. Further, let \(f_\theta(\cdot)\) denote the mapping function of a GNN model parameterized by \(\theta\), which maps a graph into the space of node labels.

**Definition 1 (Test-Time Graph Transformation (TTGT)).** TTTG requires to learn a graph transformation function \(g(\cdot)\) which refines the test graph \(G_T\) such that the pre-trained \(f_\theta\) can yield better test performance on \(g(G_T)\) than that on \(G_T\):

\[
\arg\min_g \mathcal{L}(f_\theta(g(G_T)), \mathcal{Y}_T) \quad \text{s.t.} \quad g(G_T) \in \mathcal{P}(G_T),
\]

\[
\theta^* = \arg\min_\theta \mathcal{L}(f_\theta(G_T), \mathcal{Y}_T),
\]

where \(\mathcal{L}\) denotes the loss function measuring downstream performance; \(\mathcal{P}(G_T)\) is the space of the modified graph, e.g., we may constrain the change on the graph to be small.

To optimize the TTGT problem, we are faced with two critical challenges: (1) how to parameterize and optimize the graph transformation function \(g(\cdot)\); and (2) how to formulate a surrogate loss to guide the learning process, since we do not have access to the ground-truth labels of test nodes. Therefore, we propose GTRANS and elaborate on how it addresses these challenges as follows.

### 3.1 Constructing Graph Transformation

In this subsection, we introduce how to construct the graph transformation function and detail its optimization process. Let \(G_T = \{A, X\}\) denote the test graph, where \(A \in \{0, 1\}^{N \times N}\) is the adjacency matrix, \(N\) is the number of nodes, and \(X \in \mathbb{R}^{N \times d}\) is the \(d\)-dimensional node feature matrix. Since the pre-trained GNN parameters are fixed at test time and we only care about the test graph, we drop the subscript in \(G_T\) and \(\mathcal{Y}_T\) to simplify notations in the rest of the paper.

**Construction.** We are interested in obtaining the transformed test graph \(G_T' = g(A, X) = (A', X')\). Specifically, we model feature modification as an additive function which injects perturbation to node features, i.e., \(X' = X + \Delta X\); we model the structure modification as \(A' = A \oplus \Delta A\)\footnote{\(A \oplus \Delta A\)} where \(\oplus\) stands for an element-wise exclusive OR operation and \(\Delta A \in \{0, 1\}^{N \times N}\) is a binary ma-
trix. In other words, $\Delta A_{ij} = 1$ indicates an edge flip. Formally, we seek to find $\Delta A$ and $\Delta X$ that can minimize the objective function:

$$\arg\min_{\Delta A, \Delta X} \mathcal{L}(f_\theta(A \oplus \Delta A, X + \Delta X), Y) \quad \text{s.t.} \quad (A \oplus \Delta A, X + \Delta X) \in \mathcal{P}(A, X),$$

where $\Delta A \in \{0, 1\}^{N \times N}$ and $\Delta X \in \mathbb{R}^{N \times d}$ are treated as free parameters. Further, to ensure we do not heavily violate the original graph structure, we constrain the number of changed entries in the adjacency matrix to be smaller than a budget $B$ on the graph structure, i.e., $\|\Delta A\|_2 \leq B$. We do not impose constraints on the node features to ease optimization. In this context, $\mathcal{P}$ can be viewed as constraining $\Delta A$ to a binary space as well as restricting the number of changes.

**Optimization.** Jointly optimizing $\Delta X$ and $\Delta A$ is often challenging as they depend on each other. In practice, we alternatively optimize $\Delta X$ and $\Delta A$. Notably, the optimization for $\Delta X$ is easy since the node features are continuous. The optimization for $\Delta A$ is particularly difficult in that (1) $\Delta A$ is binary and constrained; and (2) the search space of $N^2$ entries is too large especially when we are learning on large-scale graphs.

To cope with the first challenge, we relax the binary space to $[0, 1]^{N \times N}$ and then we can employ projected gradient descent (PGD) [Xu et al., 2019; Geisler et al., 2021] to update $\Delta A$:

$$\Delta A \leftarrow \Pi_P \left(\Delta A - \eta \nabla_{\Delta A} \mathcal{L}(\Delta A)\right)$$

where we first perform gradient descent with step size $\eta$ and call a projection $\Pi_P$ to project the variable to the space $\mathcal{P}$. Specifically, given an input vector $p$, $\Pi_P(\cdot)$ is expressed as:

$$\Pi_P(p) \leftarrow \begin{cases} \Pi_{[0,1]}(p), & \text{if } 1^T\Pi_{[0,1]}(p) \leq B, \\ \Pi_{[0,1]}(p - \gamma 1) \text{ with } 1^T\Pi_{[0,1]}(p - \gamma 1) = B, & \text{otherwise,} \end{cases}$$

where $\Pi_{[0,1]}(\cdot)$ clamps the input values to $[0, 1]$, $1$ stands for a vector of all ones, and $\gamma$ is obtained by solving the equation $1^T\Pi_{[0,1]}(p - \gamma 1) = B$ with the bisection method [Liu et al., 2015]. To keep the adjacency structure discrete and sparse, we view each entry in $A \oplus \Delta A$ as a Bernoulli $\sim Bernoulli(A \oplus \Delta A)$. We do

Furthermore, to enable efficient graph structure learning, it is desired to reduce the search space of updated adjacency matrix. One recent approach of graph adversarial attack [Geisler et al., 2021] proposes to sample a small block of possible entries from the adjacency matrix and update them at each iteration. This solution is still computationally intensive as it requires hundreds of steps to achieve a good performance. Instead, we constrain the search space to only the existing edges of the graph, which is typically sparse. Empirically, we observe that this simpler strategy still learns useful structure information when combined with feature modification.

### 3.2 Parameter-Free Surrogate Loss

As discussed earlier, the proposed framework GTRANS aims to improve the model generalization and robustness by learning to transform the test graph. Ideally, when we have test ground-truth labels, the problem can be readily solved by adapting the graph to result in the minimum cross entropy loss on test samples. However, as we do not have such information at test-time, it motivates us to investigate feasible surrogate losses to guide the graph transformation process. In the absence of labeled data, recently emerging self-supervised learning techniques [Xie et al., 2021; Liu et al., 2022b] have paved the way for providing self-supervision for TTGT. However, not every surrogate self-supervised task and loss is suitable for our transformation process, as some tasks are more powerful and some are weaker. To choose a suitable surrogate loss, we provide the following theorem.

**Theorem 1.** Let $\mathcal{L}_c$ denote the classification loss and $\mathcal{L}_s$ denote the surrogate loss, respectively. Let $\rho(G)$ denote the correlation between $\nabla_G \mathcal{L}_c(G, Y)$ and $\nabla_G \mathcal{L}_s(G)$, and let $\epsilon$ denote the learning rate for gradient descent. Assume that $\mathcal{L}_s$ is twice-differentiable and its Hessian matrix satisfies $\|H(G, Y)\|_2 \leq M$ for all $G$. When $\rho(G) > 0$ and $\epsilon < \frac{2\rho(G)\|\nabla_G \mathcal{L}_s(G, Y)\|_2}{M\|\nabla_G \mathcal{L}_c(G)\|_2}$, we have

$$\mathcal{L}_c(G - \epsilon \nabla_G \mathcal{L}_s(G), Y) < \mathcal{L}_c(G, Y).$$

The proof can be found in Appendix A.1. Theorem 1 suggests that when the gradients from classification loss and surrogate loss have a positive correlation, i.e., $\rho(G) > 0$, we can update the test graph by performing gradient descent with a sufficiently small learning rate such that the classification loss on the test samples is reduced. Hence, it is imperative to find a surrogate task that shares relevant information with the classification task.
Parameter-Free Surrogate Loss. As one popular self-supervised paradigm, graph contrastive learning has achieved promising performance in various graph-related tasks (Hassani & Khasahmadi, 2020; You et al., 2021; Zhu et al., 2021b), which indicates that graph contrastive learning tasks are often highly correlated with downstream tasks. This property is desirable for guiding TTGT as suggested by Theorem 1. At its core lies the contrasting scheme, where the similarity between two augmented views from the same sample is maximized, while the similarity between views from two different samples is minimized.

However, existing graph contrastive learning methods cannot be directly applied to our scenario, as they require a parameterized projection head to map augmented representations to another latent space, which inevitably alters the model architecture. To address this issue, we propose a parameter-free surrogate loss. Specifically, we apply an augmentation function \( A(\cdot) \) on input graph \( G \) and obtain an augmented graph \( \hat{A}(G) \). The node representations obtained from the two graphs are denoted as \( Z \) and \( \hat{Z} \), respectively; \( z_i \) and \( \hat{z}_i \) stand for the \( i \)-th node representation taken from \( Z \) and \( \hat{Z} \), respectively. We adopt DropEdge (Rong et al., 2020) as the augmentation function \( A(\cdot) \), and the hidden presentations \( z_i, \hat{z}_i \) are taken from the output of the second last layer of the pre-trained model. Essentially, we maximize the cosine similarity between original nodes and their augmented view while penalizing the similarity between the nodes and their negative samples:

\[
\min L_s = \sum_{i=1}^{N} (1 - \frac{z_i^\top z_i}{\|z_i\|_2}) - \sum_{i=1}^{N} (1 - \frac{\hat{z}_i^\top \hat{z}_i}{\|\hat{z}_i\|_2}), \tag{6}
\]

where \( \{z_i| i = 1, \ldots, N\} \) are the negative samples for corresponding nodes, which are generated by shuffling node features (Veličković et al., 2019). In Eq. (6), the first term encourages each node to be close while second term pushes each node away from the corresponding negative sample. Note that (1) \( L_s \) is parameter-free and does not require modification of the model architecture, or affect the pre-training process; (2) there could be other self-supervised signals for guiding the graph transformation, and we empirically compare them with the proposed surrogate loss in Appendix D.6.

Furthermore, the algorithm of GTTRANS is provided in Appendix B.

3.3 Further Analysis

In this subsection, we study the theoretical property of the proposed surrogate loss and compare the strategy of adapting data versus that of adapting model. We first demonstrate the rationality of the proposed surrogate loss through the following theorem.

Theorem 2. Assume that the augmentation function \( A(\cdot) \) generates a data view of the same class for the test nodes and the node classes are balanced. Assume for each class, the mean of the representations obtained from \( Z \) and \( \hat{Z} \) are the same. Minimizing the first term in Eq. (6) is approximately minimizing the class-conditional entropy \( H(Z|Y) \) between features \( Z \) and labels \( Y \).

The proof can be found in Appendix A.2. Theorem 2 indicates that minimizing the first term in Eq. (6) will minimize \( H(Z|Y) \), which encourages high intra-class compactness, i.e., learning a low-entrophy cluster in the embedded space for each class. However, we note that only optimizing this term can result in collapse (mapping all data points to a single point in the embedded space), which stresses the necessity of the second term in Eq. (6).

Next, we use an illustrative example to show that adapting data at test-time can be more useful than adapting model in some cases. Given test samples \( \{x_i| i = 1, \ldots, K\} \), we consider a linearized GNN \( f_0 \) which first performs aggregation through a function \( \text{Agg}(\cdot, \cdot) \) and then transforms the aggregated features via a function \( \text{Trans}(\cdot) \). Hence, only the function \( \text{Trans}(\cdot) \) is parameterized by \( \theta \).

Example. Let \( N_i \) denote the neighbors for node \( x_i \). If there exist two nodes with the same aggregated features but different labels, i.e., \( \text{Agg}(x_i, \{x_i| i \in N_1\}) = \text{Agg}(x_2, \{x_j| j \in N_2\}), y_1 \neq y_2 \), adapting the data \( \{x_i| i = 1, \ldots, K\} \) can achieve lower classification error than adapting the model \( f_0 \) at test stage.

Illustration. Let \( \tilde{x}_1 = \text{Agg}(x_1, \{x_i| i \in N_1\}) \) and \( \tilde{x}_2 = \text{Agg}(x_2, \{x_j| j \in N_2\}) \). For simplicity, we consider the following mean square loss as the classification error:

\[
\ell = \frac{1}{2} (\text{Trans}(\tilde{x}_1) - y_1)^2 + (\text{Trans}(\tilde{x}_2) - y_2)^2. \tag{7}
\]
It is easy to see that \( \ell \) reaches its minimum when \( \text{Trans}(\bar{x}_1) = y_1 \) and \( \text{Trans}(\bar{x}_2) = y_2 \). In this context, it is impossible to find \( \theta \) such that \( \text{Trans}(\cdot) \) can map \( x_1, x_2 \) to different labels since it is not a one-to-many function. However, since \( y_1 \) and \( y_2 \) are in the label space of training data, we can always modify the test graph to obtain newly aggregated features \( \bar{x}_1', \bar{x}_2' \) such that \( \text{Trans}(\bar{x}_1') = y_1 \) and \( \text{Trans}(\bar{x}_2') = y_2 \), which minimizes \( \ell \). In the extreme case, we may drop all node connections for the two nodes, and let \( x_1' \leftarrow \bar{x}_1' \) and \( x_2' \leftarrow \bar{x}_2' \) where \( \bar{x}_1' \) and \( \bar{x}_2' \) are the aggregated features taken from the training set. Hence, adapting data can achieve lower classification loss.

Remark 1. Note that the existence of two nodes with the same aggregated features but different labels is not rare when considering adversarial attack or abnormal features. We provide a figurative example in Figure 5 in Appendix A.2, where the attacker injects one adversarial edge into the graph and changes the aggregated features \( \bar{x}_1 \) and \( \bar{x}_2 \) to be the same.

Remark 2. When we consider \( \bar{x}_1 \neq \bar{x}_2, y_1 \neq y_2 \), whether we can find \( \theta \) satisfying \( \text{Trans}(\bar{x}_1) = y_1 \) and \( \text{Trans}(\bar{x}_2) = y_2 \) depends on the expressiveness of the transformation function. If it is not powerful enough (e.g., an under-parameterized neural network), it could fail to map different data points to different labels. On the contrary, adapting the data does not suffer this problem as we can always modify the test graph to satisfy \( \text{Trans}(\bar{x}_1') = y_1 \) and \( \text{Trans}(\bar{x}_2') = y_2 \).

Remark 3. The above discussion can be easily extended to nonlinear GNN by considering \( \bar{x}_1, \bar{x}_2 \) as the output before the last linear layer of GNN.

4 Experiment

4.1 Generalization on Out-of-Distribution Data

Setup. Following the settings in EERM (Wu et al., 2022a), which is designed for node-level tasks on OOD data, we validate GTRANS on three types of distribution shifts with six benchmark datasets: (1) artificial transformation for Cora (Yang et al., 2016) and Amazon-Photo (Stichur et al., 2019), (2) cross-domain transfers for Twitch-E and FB-100 (Rozemberczki et al., 2021) (Lim et al., 2021), and (3) temporal evolution for Elliptic (Pareja et al., 2020) and OGB-Arxiv (Hu et al., 2020). Moreover, Cora and Amazon-Photo have 1/1/8 graphs for training/validation/test sets. The splits are 1/1/5 on Twitch-E, 3/2/3 on FB-100, 5/5/33 on Elliptic, and 1/1/3 on OGB-Arxiv. More details on the datasets are provided in Appendix A.2. We compare GTRANS with four baselines: empirical risk minimization (ERM), i.e., standard training, data augmentation technique DropEdge (Rong et al., 2020), test-time-training method Tent (Wang et al., 2021), and the recent SOTA method EERM (Wu et al., 2022a) which is exclusively developed for graph OOD issue. Note that SR-GNN (Zhu et al., 2021a) is not included as a baseline because it specifically targets distribution shifts between the selection of training and testing nodes instead of the general OOD issue. By contrast, both Tent and EERM are designed to handle general distribution shifts. Further, we evaluate all the methods with four popular GNN backbones including GCN (Kipf & Welling, 2016a), GraphSAGE (Hamilton et al., 2017), GAT (Velickovic et al., 2018), and GPR (Chien et al., 2021). Their default setup follows that in EERM. We refer the readers to Appendix A.2 for more implementation details of baselines and GTRANS. Notably, all experiments in this paper are repeated 10 times with different random seeds.

Results. Table 1 reports the averaged performance over the test graphs for each dataset as well as the averaged rank of each algorithm. From the table, we make the following observations:

(a) Overall Performance. The proposed framework consistently achieves strong performance across the datasets: GTRANS achieves average ranks of 1.0, 1.7, 2.0 and 1.7 with GCN, SAGE, GAT and GPR, respectively, while the corresponding ranks for the best baseline EERM are 2.9, 3.4, 3.0 and 2.0. Furthermore, in most of the cases, GTRANS significantly improves the vanilla baseline (ERM) by a large margin. Particularly, when using GCN as backbone, GTRANS outperforms ERM by 3.1%, 5.0% and 2.0% on Cora, Elliptic and OGB-Arxiv, respectively. These results demonstrate the effectiveness of GTRANS in tackling diverse types of distribution shifts.

(b) Comparison to other baselines. Both DropEdge and EERM modify the training process to improve model generalization. Nonetheless, they are less effective than GTRANS, as GTRANS takes advantage of the information from test graphs. As a test-time training method, Tent also performs
Table 1: Average classification performance (%) on the test graphs. Rank indicates the average rank of each algorithm for each backbone. OOM indicates out-of-memory error on 32 GB GPU memory. The proposed GTRANS consistently ranks the best compared with the baselines.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Method</th>
<th>Amz-Photo</th>
<th>Cora</th>
<th>Elliptic</th>
<th>FB-100</th>
<th>OGB-Arxiv</th>
<th>Twitch-E</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>ERM</td>
<td>93.79±0.97</td>
<td>91.59±1.44</td>
<td>50.90±1.51</td>
<td>54.04±0.94</td>
<td>38.59±1.35</td>
<td>59.89±0.50</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>DropEdge</td>
<td>92.11±0.31</td>
<td>81.01±1.33</td>
<td>53.96±4.91</td>
<td>53.00±0.50</td>
<td>41.26±0.92</td>
<td>59.55±0.39</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td>Tent</td>
<td>94.03±1.07</td>
<td>91.87±1.36</td>
<td>51.71±2.00</td>
<td>54.16±1.00</td>
<td>39.33±1.40</td>
<td>59.46±0.55</td>
<td>3.3</td>
</tr>
<tr>
<td>EERM</td>
<td></td>
<td>94.05±0.40</td>
<td>87.21±1.53</td>
<td>53.96±0.65</td>
<td>54.24±0.55</td>
<td>OOM</td>
<td>59.35±0.85</td>
<td>2.9</td>
</tr>
<tr>
<td>GTRANS</td>
<td></td>
<td>94.13±0.77</td>
<td>94.66±0.63</td>
<td>55.88±3.10</td>
<td>54.32±0.60</td>
<td>41.59±1.20</td>
<td>60.42±0.86</td>
<td>1.0</td>
</tr>
<tr>
<td>SAGE</td>
<td>ERM</td>
<td>95.09±0.60</td>
<td>99.67±0.14</td>
<td>56.12±4.47</td>
<td>54.70±0.47</td>
<td>54.24±0.55</td>
<td>59.85±0.85</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td>DropEdge</td>
<td>92.61±0.56</td>
<td>95.85±0.30</td>
<td>52.38±3.11</td>
<td>54.51±0.69</td>
<td>38.89±1.74</td>
<td>62.14±0.12</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>Tent</td>
<td>95.72±0.43</td>
<td>99.80±0.10</td>
<td>55.89±4.87</td>
<td>54.86±0.34</td>
<td>39.58±1.26</td>
<td>62.09±0.09</td>
<td>2.3</td>
</tr>
<tr>
<td>EERM</td>
<td></td>
<td>95.57±1.32</td>
<td>98.77±0.14</td>
<td>58.20±3.55</td>
<td>53.30±0.77</td>
<td>OOM</td>
<td>62.11±0.12</td>
<td>3.4</td>
</tr>
<tr>
<td>GTRANS</td>
<td></td>
<td>96.91±0.68</td>
<td>99.45±0.13</td>
<td>60.81±5.19</td>
<td>54.64±0.62</td>
<td>43.76±1.25</td>
<td>62.15±0.13</td>
<td>1.7</td>
</tr>
<tr>
<td>GAT</td>
<td>ERM</td>
<td>96.30±0.79</td>
<td>94.81±1.28</td>
<td>65.36±2.70</td>
<td>51.77±1.41</td>
<td>40.63±1.57</td>
<td>58.53±1.00</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>DropEdge</td>
<td>90.70±0.29</td>
<td>76.91±1.53</td>
<td>63.78±2.39</td>
<td>52.65±0.88</td>
<td>42.48±0.93</td>
<td>58.89±1.01</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>Tent</td>
<td>95.99±0.46</td>
<td>95.91±1.14</td>
<td>60.07±1.66</td>
<td>51.47±1.70</td>
<td>40.06±1.19</td>
<td>58.33±1.18</td>
<td>3.3</td>
</tr>
<tr>
<td>EERM</td>
<td></td>
<td>95.57±1.32</td>
<td>98.00±0.96</td>
<td>58.14±4.71</td>
<td>53.30±0.77</td>
<td>OOM</td>
<td>62.11±0.12</td>
<td>3.4</td>
</tr>
<tr>
<td>GTRANS</td>
<td></td>
<td>96.67±0.74</td>
<td>96.37±1.00</td>
<td>66.43±2.57</td>
<td>51.16±1.72</td>
<td>43.76±1.25</td>
<td>58.59±1.07</td>
<td>2.0</td>
</tr>
<tr>
<td>GPR</td>
<td>ERM</td>
<td>91.87±0.65</td>
<td>93.00±2.17</td>
<td>64.59±3.52</td>
<td>54.51±0.33</td>
<td>44.38±0.59</td>
<td>59.72±0.40</td>
<td>2.7</td>
</tr>
<tr>
<td></td>
<td>DropEdge</td>
<td>88.81±1.48</td>
<td>79.27±1.39</td>
<td>61.02±1.78</td>
<td>55.04±0.33</td>
<td>43.65±0.77</td>
<td>59.33±0.05</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>Tent</td>
<td>97.78±0.52</td>
<td>88.82±3.10</td>
<td>67.27±0.98</td>
<td>55.95±0.03</td>
<td>OOM</td>
<td>61.57±0.12</td>
<td>2.0</td>
</tr>
<tr>
<td>EERM</td>
<td></td>
<td>90.78±0.52</td>
<td>93.05±2.02</td>
<td>69.03±2.33</td>
<td>54.38±0.31</td>
<td>46.00±0.46</td>
<td>60.14±0.53</td>
<td>1.7</td>
</tr>
<tr>
<td>GTRANS</td>
<td></td>
<td>91.93±0.73</td>
<td>93.05±2.02</td>
<td>69.03±2.33</td>
<td>54.38±0.31</td>
<td>46.00±0.46</td>
<td>60.14±0.53</td>
<td>1.7</td>
</tr>
</tbody>
</table>

3 Tent cannot be applied to models which do not contain batch normalization layers.

Figure 2: Results on Cora under OOD. GTRANS improves GCN on most test graphs.

Table 2: Efficiency comparison. GTRANS is more time- and memory-efficient than EERM.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Method</th>
<th>Extra Running Time (s)</th>
<th>Total GPU Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cora Photo Ellip. Arxiv</td>
<td>Cora Photo Ellip. Arxiv</td>
</tr>
<tr>
<td>EERM</td>
<td></td>
<td>25.9 396.4 607.9 -</td>
<td>2.5 10.5 12.8 -</td>
</tr>
<tr>
<td>GTRANS</td>
<td>0.3</td>
<td>1.4 1.5 1.3 3.9</td>
<td></td>
</tr>
</tbody>
</table>

4.2 Robustness to Abnormal Features

Setup. Following the setup in AirGNN [Liu et al., 2021a], we evaluate the robustness in the case of abnormal features. Specifically, we simulate abnormal features by assigning random features taken from a multivariate standard Gaussian distribution to a portion of randomly selected test nodes. Note that the abnormal features are injected after model training (at test time) and we vary the ratio of noisy nodes from 0.1 to 0.4 with a step size of 0.05. This process is performed for four datasets: the original version of Cora, Citeseer, Pubmed, and OGB-Arxiv. In these four datasets, the training graph and the test graph have the same graph structure but the node features are different. Hence, we use the training classification loss combined with the proposed contrastive loss to optimize GTRANS. We use GCN as the backbone model and adopt four GNNs as the baselines including GAT [Veličković et al., 2018], APPNP [Klicpera et al., 2018], AirGNN and AirGNN-t.
Note that AirGNN-t tunes the message-passing hyper-parameter in AirGNN at test time. For a fair comparison, we tune AirGNN-t based on the performance on both training and validation nodes.

Results. For each model, we present the node classification accuracy on both abnormal nodes and all test nodes (i.e., both normal and abnormal ones) in Figure 3 and Figure 7 (See Appendix D.2), respectively. From these figures, we have two observations. First, GT\textsuperscript{RANS} significantly improves GCN in terms of the performance on abnormal nodes and all test nodes for all datasets across all noise ratios. For example, on Cora with 30% noisy nodes, GT\textsuperscript{RANS} improves GCN by 48.2% on abnormal nodes and 31.0% on overall test accuracy. This demonstrates the effectiveness of the graph transformation process in GT\textsuperscript{RANS} in alleviating the effect of abnormal features. Second, GT\textsuperscript{RANS} shows comparable or better performance with AirGNNs, which are the SOTA defense methods for tackling abnormal features. It is worth mentioning that AirGNN-t improves AirGNN by tuning its hyper-parameter at test time, which aligns with our motivation that test-time adaptation can enhance model test performance. To further understand the effect of graph transformation, we provide the visualization of the test node embeddings obtained from abnormal graph (0.3 noise ratio) and transformed graph for Cora in Figures 4a and 4b, respectively. We observe that the transformed graph results in well-clustered node representations, which indicates that GT\textsuperscript{RANS} can promote intra-class compactness and counteract the effect of abnormal patterns.

4.3 **Robustness to Adversarial Attack**

Setup. We further evaluate GT\textsuperscript{RANS} under the setting of adversarial attack where we perturb the test graph, i.e., evasion attack. Specifically, we use PR-BCD (Geisler et al., 2021), a scalable attack method, to attack the test graph in OGB-Arxiv. We focus on structural attacks, and vary the perturbation rate, i.e., the ratio of changed edges, from 5% to 25% with a step of 5%. Similar to Section 4.2, we adopt the training classification loss together with the proposed contrastive loss to optimize GT\textsuperscript{RANS}. We use GCN as the backbone and employ four robust baselines including GAT (Veličković et al., 2018), RobustGCN (Zhu et al., 2019), SimPGCN (Jin et al., 2021a) and GCNJaccard (Xu et al., 2019) as comparisons. Among them, GCNJaccard pre-processes the attacked graph by removing edges where the similarities of connected nodes are less than a threshold; we tune this threshold at test time based on the performance on both training and validation nodes.

Results. Table 3 reports the performances under structural evasion attack. We observe that GT\textsuperscript{RANS} consistently improves the performance of GCN under different perturbation rates of adversarial attack. Particularly, GT\textsuperscript{RANS} improves GCN by a larger margin when the perturbation rate is higher. For example, GT\textsuperscript{RANS} outperforms GCN by over 40% under the 25% perturbation rate. Such observation suggests that GT\textsuperscript{RANS} can counteract the devastating effect of adversarial attacks. In addition, the best performing baseline GCNJaccard also modifies the graph at test time, which demonstrates the importance of test-time graph adaptation. Nonetheless, it consistently underperforms our
Table 3: Node classification accuracy (%) under different perturbation (Ptb.) rates of structure attack.

<table>
<thead>
<tr>
<th>Ptb. Rate</th>
<th>GCN</th>
<th>GAT</th>
<th>RobustGCN</th>
<th>SimPGCN</th>
<th>GCNJaccard</th>
<th>GTRANS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>57.47±0.54</td>
<td>64.66±0.43</td>
<td>61.55±1.20</td>
<td>61.30±0.42</td>
<td>65.01±0.26</td>
<td>66.29±0.25</td>
</tr>
<tr>
<td>10%</td>
<td>47.97±0.65</td>
<td>61.20±0.70</td>
<td>58.15±1.55</td>
<td>57.01±0.70</td>
<td>63.25±0.30</td>
<td>65.16±0.52</td>
</tr>
<tr>
<td>15%</td>
<td>38.04±1.22</td>
<td>58.96±0.59</td>
<td>55.91±1.27</td>
<td>54.13±0.73</td>
<td>61.83±0.29</td>
<td>64.40±0.38</td>
</tr>
<tr>
<td>20%</td>
<td>29.05±0.73</td>
<td>57.29±0.49</td>
<td>54.39±1.09</td>
<td>52.26±0.87</td>
<td>60.57±0.34</td>
<td>63.44±0.50</td>
</tr>
<tr>
<td>25%</td>
<td>19.58±2.32</td>
<td>55.86±0.53</td>
<td>52.76±1.44</td>
<td>50.46±0.85</td>
<td>59.17±0.39</td>
<td>62.95±0.67</td>
</tr>
</tbody>
</table>

proposed GTRANS, indicating that a learnable transformation function is needed to achieve better robustness under adversarial attacks, which GCNJaccard does not employ.

**Interpretation.** To understand the modifications made on the graph, we compare several properties among clean graph, attacked graph (20% perturbation rate), graph obtained by GCNJaccard, and graph obtained by GTRANS in Table 7 in Appendix D.3. First, adversarial attack decreases homophily and feature similarity, but GTRANS and GCNJaccard promote such information to alleviate the adversarial patterns. Our experiment also shows that GTRANS removes 77% adversarial edges while removing 30% existing edges from the attacked graph. Second, both GTRANS and GCNJaccard focus on deleting edges from the attacked graph, but GCNJaccard removes a substantially larger amount of edges, which may destroy clean graph structure and lead to sub-optimal performance.

### 4.4 Further Analysis

**Cross-Architecture Transferability.** Since the outcome of GTRANS is a refined graph, it can conceptually be employed by any GNN model. In other words, we can transform the graph based on one pre-trained GNN and test the transformed graph on another pre-trained GNN. To examine such transferability, we perform experiments on four GNNs including GCN, APPNP, AirGNN and GAT under the abnormal feature setting with 30% noisy nodes on Cora. The results on all test nodes in Table 4. Note that “Tr” stands for GNNs used in TTGT while “Te” denotes GNNs used for obtaining final predictions on the transformed graph; “Noisy” indicates the performance on the noisy graph. We observe that the transformed graph yields good performance even outside the scope it was optimized for. We anticipate that such transferability can alleviate the need for costly re-training on new GNNs.

**Adapting Model vs. Adapting Data.** We empirically compare the performance between adapting data and adapting model and consider the OOD and abnormal feature settings. Specifically, we use GCN as the backbone and adapt the model parameters by optimizing the same loss function as used in GTRANS. The results are shown in Figure 4c and 4d. In OOD setting, both adapting model and adapting data can generally improve GCN’s performance. Since their performances are still close, it is hard to give a definite answer on which strategy is better. However, we can observe significant performance differences when the graph contains abnormal features: adapting data outperforms adapting model on 3 out of 4 datasets. This suggests that adapting data can be more powerful when the data is perturbed, which aligns with our analysis in Section 3.3.

**Effect of Surrogate Loss.** Recall that we used a combined loss of contrastive loss and training loss in the settings of abnormal features and adversarial attack. We now examine the effect of using them alone and show the results in Tables 8 and 9 in Appendix D.4. We observe that (1) both of them bring improvement over vanilla GCN, while optimizing the training loss alone improves more than optimizing the other; and (2) combining them always yields a better or comparable performance.

**Learning Features vs. Learning Structure.** Since our framework learns both node features and graph structure, we investigate when one component plays a more important role than the other. Our results are shown in Tables 10 and 11 in Appendix D.5. From the tables, we observe that (1) while each component can improve the vanilla performance, feature learning is more crucial for counteracting feature corruption and structure learning is more important for defending structure corruption; and (2) combining them generally yields a better or comparable performance.
5 Conclusion

GNNs tend to yield unsatisfying performance when the presented data is sub-optimal. To tackle this issue, we seek to enhance GNNs from a data-centric perspective by transforming the graph data at test time. We propose GTRANS which optimizes a contrastive surrogate loss to transform graph structure and node features, and provide theoretical analysis with deeper discussion to understand this framework. Experimental results on distribution shift, abnormal features and adversarial attack have demonstrated the effectiveness of our method. In the future, we plan to explore more applications of our framework such as mitigating degree bias and long-range dependency.

References


Jintang Li, Tao Xie, Chen Liang, Fenfang Xie, Xiangnan He, and Zibin Zheng. Adversarial attack on large scale graph. *IEEE Transactions on Knowledge and Data Engineering*, 2021.


Qi Zhu, Natalia Ponomareva, Jiawei Han, and Bryan Perozzi. Shift-robust gns: Overcoming the limitations of localized graph training data. *Advances in Neural Information Processing Systems*, 34, 2021a.

